كلية علوم الحاسب والمعلومات
قسم هندسة البرمجيات

King Saud University

College of Computer and Information Sciences

Department of Software Engineering

SWE 486 – Cloud Computing and Big Data

Expo 2020 Dubai

Sentiment Analysis

PHASE 3

[Project GitHub Repository]

#	NAME	ID	SECTION
1	Dalal Bin Humaid		68348
2	Muneerah Alsunaidi		54978
3	Reem Aldosari		54979
4	Sarah alsuqair		68348
5	Shahad Alshahrani		54978
6	Safia Assiri		54978

GROUP #	8
SUPERVISOR	Hailah Almazrua

Submission Date: April 28, 2022

Table of Contents

List of Tables

TABLE OF FIGURES

PART 3

SENTIMENT ANALYSIS

The purpose of the sentiment analysis of the dataset is to find out what users think about the expo generally, the meaning of sentiment analysis is the systematic identification, extraction, quantification, and study of emotional states and subjective information using natural language processing, text analysis, and computational linguistics, and biometrics [1].

We applied sentiment analysis by using Mazajak, which is an Online Arabic Sentiment Analyzer that assigns a three-way sentiment categorization to one of the following classifications (Positive, Negative, and Neutral) [2].

LABELING THE DATASET

We first imported the API from Mzajak's website

```python
# mazajak_api.py > ...
import requests
import json

'''
This function offers the ability to predict the sentiment of a single sentence
through the API, the sentiment is one of three classes (positive negative, neutral)
Input:
        sentence(str): the input sentence of which the sentiment is to be predicted
Output:
        prediction(str): the sentiment of the given sentence
'''

def predict(sentence):
    url = "http://mazajak.inf.ed.ac.uk:8000/api/predict"
    to_sent = {'data': sentence}
    data = json.dumps(to_sent)
    headers = {'content-type': 'application/json'}
    # sending get request and saving the response as response object
    response = requests.post(url=url, data=data, headers=headers)

    prediction = json.loads(response.content)['data']

    return prediction


'''
This function offers the ability to predict the sentiment of a list of sentences
through the API, the sentiment is one of three classes (positive negative, neutral)
Input:
        sent_lst(list of str): the input list of which the sentiment of its sentences is to be predicted
Output:
        prediction(list of str): the sentiments of the given sentences
'''

def predict_list(sent_lst):
    url = "http://mazajak.inf.ed.ac.uk:8000/api/predict_list"
    to_sent = {'data': sent_lst}
    data = json.dumps(to_sent)
    headers = {'content-type': 'application/json'}
    # sending get request and saving the response as response object
    response = requests.post(url=url, data=data, headers=headers)

    prediction = json.loads(response.content)['data']

    return prediction

```

Figure 1 Mazajak's API

Then, we imported the needed function which in our case was *predict()* the following code is how we imported it as well as reading our dataset and storing it in a data frame.

```
import pandas as pd
from mazajak_api import predict
```
```
negative
positive
['negative', 'positive']
```

Figure 2 importing the API

```
tweets = pd.read_csv('Tweets\\final_tweets_cleaned.csv')
tweets.columns
```
```
Index(['Unnamed: 0', 'ID', 'Tweet', 'Timestamp', 'Likes', 'Retweets', 'Length',
       'Date', 'Time'],
      dtype='object')
```
```
tweets = tweets.drop(['Unnamed: 0'], axis=1)
```

Figure 3 reading the cleaned tweets

Since the API is based on http get requests, we faced a problem trying to apply the function to the entire dataset. When a certain time interval elapses the API closes the connection which results in an error. We managed to bypass this by splitting the data frame into chunks that can be processed faster. Each time a chunk gets processed the connection is closed and opened again. It still took a lot of time classifying the entire dataset the following code snippet illustrates the function used and the time it took.

```
size = 50
chunks = [tweets[i:i+size].copy() for i in range(0, tweets.shape[0], size)]
```
```
no_chunks = len(chunks)
no_chunks
```
```
160
```

Figure 4 splitting the data frame into chunks

```
for i in range(0, no_chunks):
    chunks[i]['Class'] = chunks[i]['Tweet'].map(predict)
    print(f'_________ lap {i+1} of {no_chunks} _________')
```

```
[25]  ✓ 631m 19.1s
```

```
Output exceeds the size limit. Open the full output data in a text editor
_________ lap 1 of 160 _________
_________ lap 2 of 160 _________
_________ lap 3 of 160 _________
_________ lap 4 of 160 _________
```

Figure 5 running the function on each chunk

Now we can merge the chunks back to a single data frame, display the output and store it in a .csv file

```
tweets = pd.concat(chunks)
```

```
✓ 2.1s
```

Figure 6 merge chunks into one data frame

	ID	Tweet	Timestamp	Likes	Retweets	Length	Date	Time	Class
7970	1510697936317030406	اغلق معرض اكسبو الامارات اجلته الجاءحة ابوابه الترقب وسبعة مليارات الاستثمارات و ساعة العمل وستة اشهر الاحتفالات الحدث الارث الحقيقي متوقعا الاجابة عبر السلسلة	2022-04-03 19:17:10+00:00	9	1	279	2022-04-03	19:17:10	neutral
7971	1510688181640962050	اختام معرض اكسبو دبي يومين الخميس فرصة المعرض فترة سرني شاهدته حسن تنظيم وتظاهره عالمية ناجحة نبارك لدولة الامارات الشقيقة النجاح المتفرد وباذن اله نامل استضافة معرض اكسبو الدولي الرياض تحقيق رءية السعودية تصويري	2022-04-03 18:38:24+00:00	1	0	304	2022-04-03	18:38:24	positive
7972	1510686349417984006	هاشتاق عربي انتهاءه الامارات تكشف عد زوار اكسبو دبي	2022-04-03 18:31:07+00:00	0	0	112	2022-04-03	18:31:07	neutral
7973	1510686265649340420	الحمدله انجاز راءع شكرا الامارات الحبيبه نبارك نجاح اكسبو دبي ونجاح جناح السعودية في اكسبو ونءكد نجاح الامارات نجاحنا	2022-04-03 18:30:47+00:00	0	2	160	2022-04-03	18:30:47	positive
7974	1510686095054356487	اكسبو ابرز الارقام القياسية حقها جناح السعودية اكسبو دبي	2022-04-03 18:30:06+00:00	0	0	100	2022-04-03	18:30:06	positive

Figure 7 displaying the output

```
tweets.to_csv('Tweets\\tweets_classified.csv')
```

```
✓ 2.1s
```

Figure 8 storing the tweets

Since we are basing our labeling off another model, there is still room for error and misclassification. Once we got Mazajak's model to label the data we went through and double-checked each tweet and updated the ones we saw was incorrectly classified.

Descriptive Analysis

Descriptive Analysis is the type of analysis of data that helps describe, show, or summarize data points in a constructive way such that patterns might emerge that fulfill every condition of the data [4]. Since we have 5875 tweets about expo, we applied descriptive analysis on it to view some basic statistical details like percentile, mean, variance, standard deviation of a data frame.

Implementation

Import important libraries

```
In [78]: import pandas as pd
         import csv
```

Figure 9 import libraries

In the figure 10 the result after used describe () method which is helping to view some basic statistical details like percentile, mean, std, min, max of a data frame.

```
[5]: d_tweets.describe()
```

[5]:

	Column1	index	ID	Likes	Retweets	Length
count	5875.000000	5875.000000	5.875000e+03	5875.000000	5875.000000	5875.000000
mean	3903.186383	4051.054468	1.504576e+18	6.618043	2.430809	166.516085
std	2345.432024	2194.516953	5.170023e+15	37.574642	14.314828	77.776763
min	0.000000	0.000000	1.494300e+18	0.000000	0.000000	16.000000
25%	1468.500000	2246.000000	1.500010e+18	0.000000	0.000000	104.000000
50%	4244.000000	4245.000000	1.506410e+18	1.000000	0.000000	158.000000
75%	5988.500000	5989.500000	1.509650e+18	3.000000	1.000000	229.000000
max	7457.000000	7458.000000	1.511840e+18	1362.000000	471.000000	341.000000

Figure 10 view statical details

Read tweets from the file classified.csv:

```
In [22]: analysis_tweets = pd.read_csv('final_tweets_classified.csv')
```

Figure 11 read file

Retrieve the Shape & column of the data frame:

```
In [23]: analysis_tweets.shape
Out[23]: (5875, 11)

In [24]: analysis_tweets.columns
Out[24]: Index(['Column1', 'index', 'ID', 'Tweet', 'Timestamp', 'Likes', 'Retweets',
               'Length', 'Date', 'Time', 'Class'],
              dtype='object')
```

Figure 12 method shape and columns

The shape of the data frame has 5875 rows and 11 columns (‘Tweet , ‘Class’)

Data Frame summary:

```
In [25]: #summary of data frame
         analysis_tweets.info(verbose=True)

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 5875 entries, 0 to 5874
         Data columns (total 11 columns):
          #   Column     Non-Null Count  Dtype
         ---  ------     --------------  -----
          0   Column1    5875 non-null   int64
          1   index      5875 non-null   int64
          2   ID         5875 non-null   float64
          3   Tweet      5875 non-null   object
          4   Timestamp  5875 non-null   object
          5   Likes      5875 non-null   int64
          6   Retweets   5875 non-null   int64
          7   Length     5875 non-null   int64
          8   Date       5875 non-null   object
          9   Time       5875 non-null   object
          10  Class      5875 non-null   object
         dtypes: float64(1), int64(5), object(5)
         memory usage: 505.0+ KB
```

Figure 13 method info

We used .info() function from pandas to display a summary of Dataframe that contains number dtypes and columns number and info .

Peak with head (n)

The .head(8) function from pandas that calls the first 8 rows for the dataframe with taking the order into account . We use it for better testing performance to know if the data frame has the right type or not .

```
In [26]: #check if the data frame has the right type
         analysis_tweets.head(8)
```

Out[26]:

	Column1	index	ID	Tweet	Timestamp	Likes	Retweets	Length	Date	Time	Class
0	0	0	1497300000000000000.00000	قبيل الاحتفال بـ اليوم الدولي لمراة تستعد كوكب...	2/25/22 23:00	1	0	198	2/25/2022	8:00:01 PM	positive
1	1	1	1497300000000000000.00000	لحظات نزول دابانغ سلمان خان علا ارض المسرح اله...	2/25/22 22:57	2	0	207	2/25/2022	7:57:14 PM	positive
2	2	2	1497300000000000000.00000	ليلة ميزه بانتظارنا الحانها شرقية نجومها است...	2/25/22 22:47	0	0	232	2/25/2022	7:47:38 PM	positive
3	3	3	1497300000000000000.00000	اصدق حفلة احلا فستان مستان انا بختار واذا حفل...	2/25/22 22:40	3	1	168	2/25/2022	7:40:37 PM	positive
4	4	4	1497290000000000000.00000	جنون بعده جنون دابانغ سلمان خان يستعرض حشد كب...	2/25/22 22:21	5	2	178	2/25/2022	7:21:09 PM	positive
5	5	5	1497290000000000000.00000	اوركسترا الفردوس تمتع زوار اكسبو دبي بعزفها ل...	2/25/22 22:17	2	0	163	2/25/2022	7:17:27 PM	positive
6	6	6	1497290000000000000.00000	اكسبو اكسبو دبي الامارات جناح سلطنة عمان ارق...	2/25/22 22:13	4	1	147	2/25/2022	7:13:25 PM	positive
7	7	7	1497280000000000000.00000	ليلة حتكون مليءه باحلا الانغام الشرقية كونوا ...	2/25/22 21:55	7	2	251	2/25/2022	6:55:27 PM	positive

Figure 14 method head(n)

our data is complex and have a lot of information so we separate the likes and the retweets from our data but since we could count the likes and retweets , we made it as two .csv files .

Statistics.

The .describe() function from pandas calculates the mean, std and IQR values. It excludes character columns and calculates summary statistics only for numeric columns.

```
In [27]: #statistic of charachter column
         analysis_tweets.describe(include=['object'])
```

Out[27]:

	Tweet	Timestamp	Date	Time	Class
count	5875	5875	5875	5875	5875
unique	5875	4922	39	5389	3
top	قبيل الاحتفال بـ اليوم الدولي لمراة تستعد كوكب...	3/31/22 21:59	4/1/2022	11:00:00 AM	positive
freq	1	12	700	12	3189

Figure 15 method .describe() to show statistics information .

```
In [28]: #calculate mean of the retweets column
         analysis_tweets.loc[:,"Retweets"].mean()

Out[28]: 2.4308085106382977

In [29]: #calculate mean of the likes column
         analysis_tweets.loc[:,"Likes"].mean()

Out[29]: 6.618042553191489

In [30]: #compute the variance of the data frame
         analysis_tweets.var(numeric_only=True)

Out[30]: Column1                                5501051.37741
         index                                  4815904.65791
         ID          26729139296303630565872211329024.00000
         Likes                                     1411.85374
         Retweets                                   204.91430
         Length                                    6049.22493
         dtype: float64
```

Figure 16 Counter for the data frame values and variables .

Referring to this insight, we acknowledged that we will not consider the Likes and Retweets when it comes to our judgment on the data.

```
In [31]: analysis_tweets['Class'].value_counts()

Out[31]: positive    3189
         neutral     2464
         negative     222
         Name: Class, dtype: int64
```

Figure 17 Counter for the data frame

The value counts() pandas function returns objects containing counts of unique values "sentiment". So that the first element "positive" is the most frequently-occurring element and the "negative" is the least frequently-occouring

Word occurrences

The reason we had to read a data frame only containing 2 columns ‘Tweet’ and ‘Class’ was to count the idf for getting term Frequency

```
In [32]: #importing libraries for word occurrences and count
         from sklearn.feature_extraction.text import TfidfTransformer
         from sklearn.feature_extraction.text import CountVectorizer

In [33]: #initiate the CountVectorizer
         countV=CountVectorizer()
         #generate word count for the words
         word_count=countV.fit_transform(analysis_tweets['Tweet'].values.astype('U'))
         word_count.shape

Out[33]: (5875, 19133)

In [34]: #transform count matrix to normal tf-idf
         tfidf_transform=TfidfTransformer(smooth_idf=True,use_idf=True)
         #idf values
         tfidf_transform.fit(word_count)

Out[34]: TfidfTransformer()

In [35]: #print idf values
         df_idf=pd.DataFrame(tfidf_transform.idf_,index=countV.get_feature_names(),columns=['idf_weights'])
```

Figure 18 import sklearn libraries to count the idf

Most frequent terms

```
In [36]: df_idf.sort_values(by=['idf_weights']).head(15)
Out[36]:
```

	idf_weights
اكسبو	1.22758
دبي	1.46200
علا	2.38698
الامارات	2.70135
جناح	3.15067
عينك	3.41333
العالم	3.43641
محمد	3.67228
الحدث	3.85263
داءما	3.85263
راشد	3.86749
العالمي	3.90098
شكرا	3.92289
معرض	3.92606
دولة	3.95505

Figure 19most frequent terms using sort by idf

Least frequent terms

```
In [37]: df_idf.sort_values(by=['idf_weights']).tail(15)
```

Out[37]:

	idf_weights
حيث	8.98548
حيفضل	8.98548
حين	8.98548
حينا	8.98548
حينها	8.98548
حيويا	8.98548
حكّامها	8.98548
خابوا	8.98548
خاتمة	8.98548
خاسر	8.98548
خاصة	8.98548
خاصية	8.98548
خاطرة	8.98548
خربوش	8.98548
روؤعـه	8.98548

Figure 20 least frequent terms using sort by idf

Median

```
#median of Length
analysis_tweets.loc[:,"Length"].median()
```

Figure 21 median of length

Output:

```
158.0
```

```
#median of retweets
analysis_tweets.loc[:,"Retweets"].median()
```

Figure 22 median of Retweets

Output:

```
0.0
```

```
#median of Likes
analysis_tweets.loc[:,"Likes"].median()
```

Figure 23 median of Likes

Output:

```
1.0
```

Visualize using a bar chart import matplotlib to view top 5 likes:

```
import matplotlib.pyplot as plt
# visualize the results

tweets_by_Likes = analysis_tweets['Likes'].value_counts()
fig, ax = plt.subplots()

ax.tick_params(axis='x', labelsize=15)
ax.tick_params(axis='y', labelsize=10)

ax.set_xlabel('Likes', fontsize=15)
ax.set_ylabel(' tweets' , fontsize=15)
ax.set_title('Top 5 Likes', fontsize=15, fontweight='bold')

tweets_by_Likes[:5].plot(ax=ax, kind='bar')
```

Output:

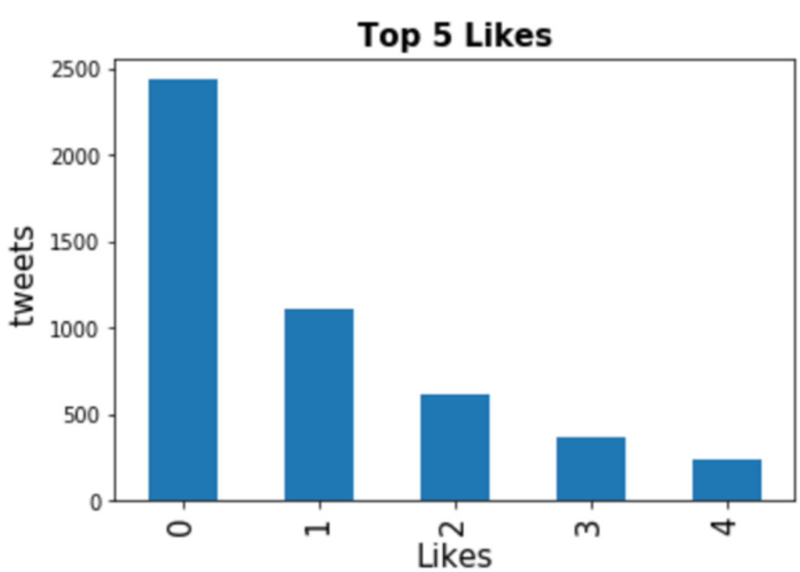


Figure 24 bar chart

Import seaborn, and matplotlib as the matplotlib is a comprehensive library and is used for creating static, animated, and interactive visualizations in Python

```
In [69]: import seaborn as sns
         import matplotlib.pyplot as plt
```

Figure 25 importing seaborn and matplotlib libraries

Visualizing using seaborn which is a library used for making statistical graphics to view the class if it’s negative or neutral or positive

```
In [70]: g = sns.FacetGrid(data=analysis_tweets, col='Class') # sperate based on Class
         g.map(plt.hist, 'Length', bins=50)
```

Figure 26 separate statistical graphics based on the Class

Output:

```
Out[39]: <seaborn.axisgrid.FacetGrid at 0x26f7261f5e0>
```

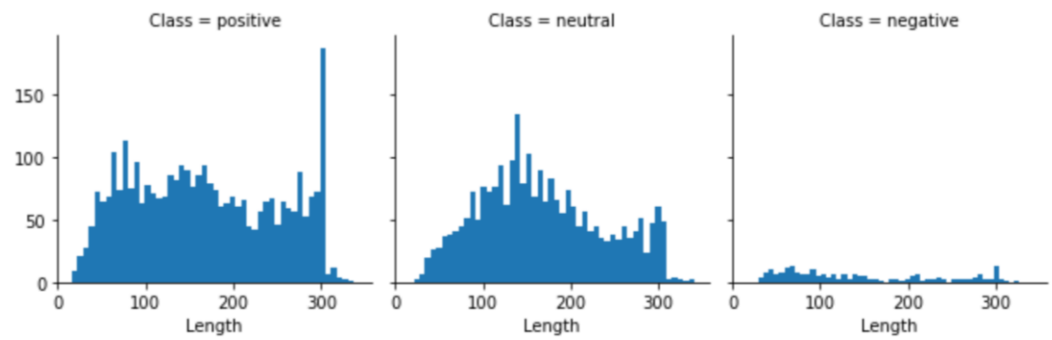


Figure 27 statistical graphics for the Class

PREDICTIVE ANALYSIS

MODEL SELECTION

We will be testing three different models, and fine tune the one with the highest scoring. There are multiple factors to consider when choosing a model. First being is the type of prediction we are concerned with. In machine learning there are three types or areas: supervised, unsupervised and reinforcement learning [5]. In our case we will be using supervised models and we had already labelled our dataset as described in the previous section. It is also crucial to note that sentiment analysis can use unsupervised learning algorithms as well.

Other factors that can affect model selection are the type of data and number of outputs/classes – some models only provide binary classification. Also, performance is a huge factor especially if the model will be deployed to its users and performance can be determined by the time it takes to train the model or the time it takes to make a prediction [6]. And most importantly how well the model performs and its accuracy.

The following is an overview of our candidate models and why they were chosen. We will then describe how we implemented and assessed their scorings.

1. Naïve Bayes

A probabilistic classifier that is based on a statistical theory known as **Baye's rule**. It is considered naïve as it does not take into consideration conditional dependence [7] . It answers the basic question of what is the probability of y, given X this is called the posterior probability of y and can be illustrated as the following formula.

$$P(y|x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n|y)}{P(x_1, \dots, x_n)}$$

Using sentiment analysis, we aim to find the answer to **what is the probability that this tweet is positive given its features or characteristics**. Naïve Bayes is used frequently for spam filtering and document categorization. Although it is fairly simple, it can produce high accuracies without the need to fine tune the model's hyperparameters.

2. Support Vector Machine

Abbreviated as SVM, it aims to find the optimal hyperplane (a decision boundary – simply put. It is a line in 2-dimensions and a plane in 3-dimension. Thus, a hyperplane in n-dimensions) that separates the data points to distinguishable classes by maximizing the margins. SVM is best explained using a simple illustration. Assume you have the following datapoints

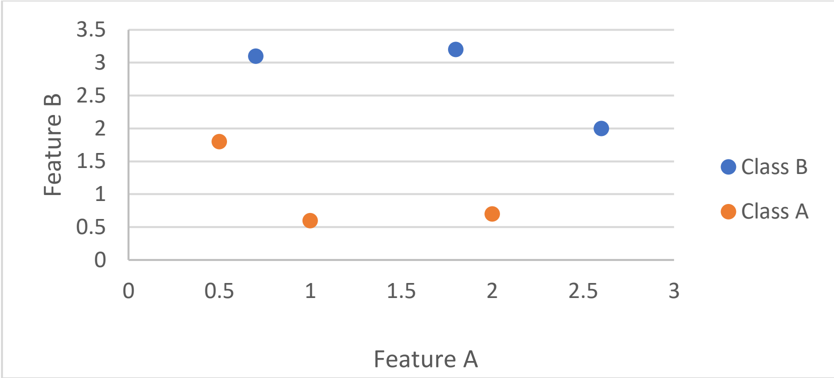


Figure 28 datapoints distribution

SVM aims to find the optimal margin that best separates the data, it does so by first finding the optimal 'line' to separate the two classes

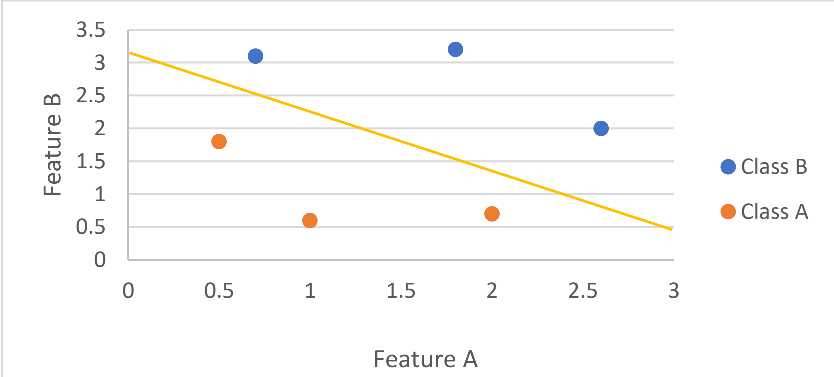


Figure 29 line of best fit

Next, it establishes the maximum margin which can be a soft margin – allows for misclassification or a hard margin – does not allow any misclassification.

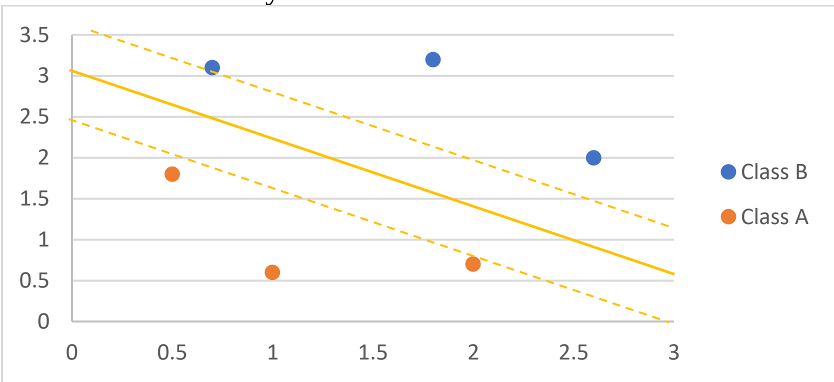


Figure 30 maximizing the margin

The points the line touches are considered **support vectors** – they are the point that influence the model's decision and form the basis of SVM. Now that we covered the basics of how SVM operates on a 2-dimensional datasets, we will explore how it scales to larger dimensions, which is where SVM shines.

The Kernel Trick

Expressed in layman terms, the kernel trick transforms a complex input space into a dimensionally higher input space and finds the hyperplane that separates that data. It then maps back to its original space. This is also best explained via an illustration.

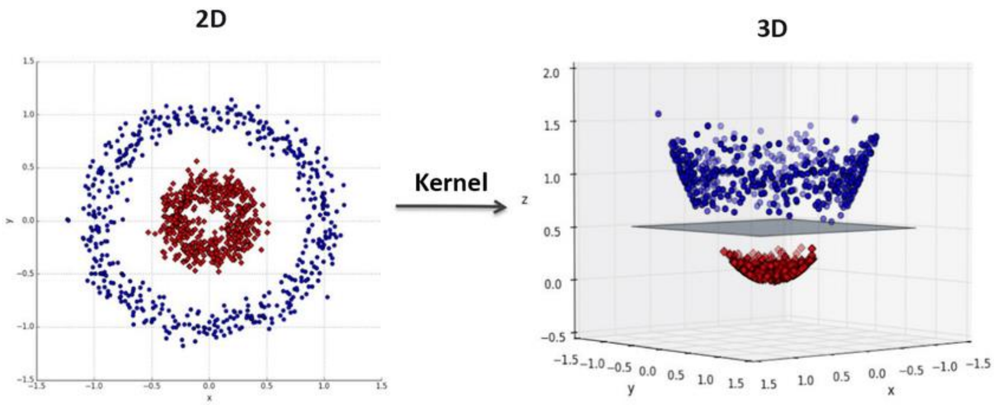


Figure 31 Kernel trick – image credit: Marouane Hachimi

In the right we can see that it is a lot harder to find the optimal margin in this 2-dimensional space. However, once we transformed the input space to a higher dimension we managed to easily find the hyperplane. *Figure 32* illustrates the how the mapping is established. The equation in the right results in a circle

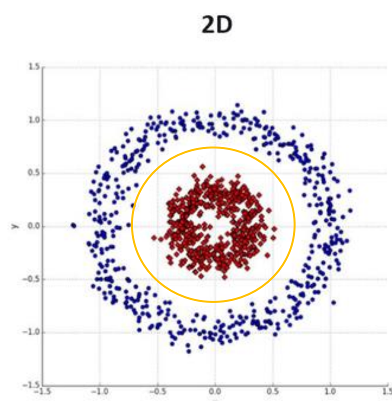


Figure 32 boundary in 2-dimensions

The kernel trick allows us to perform all of this in the original input space. i.e., we are not actually transforming computing the input space in a higher dimensions.

One of the biggest advantages SVM has is its robustness in terms of handling huge dimensions. this makes it a great candidate when dealing with textual data. It can also be interpreted easily and flexible to be used as a regression model.

3. AdaBoost

Part of the ensembles methods. Where the goal is to train on a weak learner – A classifier that is not complex and does not yield accurate results on its own. When the ensembles trains on multiple weak learners it combines them to produce the actual results. The combination can be done in two ways, by taking the **average** of all the base estimators – weak learners or by **boosting** the base estimators and assigning different weights.

AdaBoost as its name suggests is a boosting ensembles method which uses decision trees as its base estimator. It uses a type of decision tree which is called a stump. Stumps are simply decision trees with a root and one level children i.e., a decision tree with a depth of 1. *figures 33, 34* illustrate examples of valid stumps

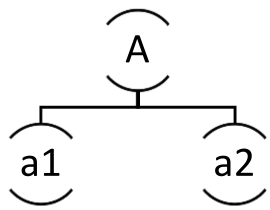


Figure 33 valid stump I

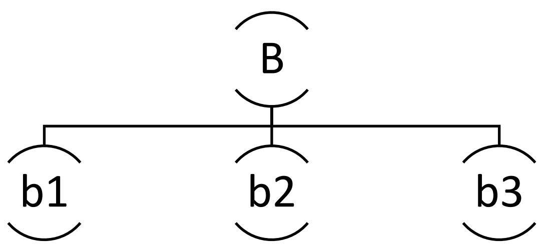


Figure 34 valid stump II

AdaBoost works sequentially by first giving each datapoint the same weight. It then starts training the classifier by creating a stump for every feature, and for each wrongly classified data point it increases its weight. Each constructed stump has a say in the final classification. How strong it sways the decision is calculated by alpha. Alpha can be calculated in different ways, the important part is the more accurate the stump is the higher alpha value it has. Therefore, the next training will take into account wrongly classified datapoints and ensures that they are accurately classified. The initial weight is given by:

$$w(p_i) = \frac{1}{N}, i = 1, 2, \dots n;\ N = \ total\ number\ of\ entries$$

AdaBoost is not prone to overfitting. It is also highly customizable since we can control the base estimator to be any classifier.

Model Training and Evaluation

Since we will be training three different classifiers and evaluate them on the same set of criteria, we have created a pipeline that does so for every classifier. Prior to that we will have to read, parse and asses the data to be trained. As explained previously our goal is to perform binary classification. Therefore, we care about positive or negative opinions only.

Feature Extraction and Preprocessing

First, we dropped the third class and reset the index.

```python
# Drop the neutral class
tweets = tweets[tweets.Class ≠ 'neutral']
tweets = tweets.reset_index()
```
✓ 0.7s Python

Figure 35 drop the neutral class

Next, we mapped each positive class to 1 and negative to 0. despite having textual data, classifiers deal with numerical representation of the data.

```python
# Convert the classes negative and positive to 0 and 1 respectively
tweets['Class'] = tweets['Class'].map({'negative': 0, 'positive': 1})
```
✓ 0.5s Python

Figure 36 convert positive and negative class labels to 1 and 0

```python
display(tweets.tail(10))
```
[8] ✓ 0.9s

	ID	Tweet	Timestamp	Likes	Retweets	Length	Date	Time	Class
3401	1.510730e+18	...الي مدبوغ مسوي انسحب و الي مشترك دورة تجارية م	4/4/22 0:33	15	1	287	4/3/2022	9:33:33 PM	0
3402	1.510720e+18	حمد ماجد عبداله معرض اكسبو دبي حدثا فريدا ميزا	4/3/22 23:58	0	0	88	4/3/2022	8:58:37 PM	1
3403	1.510720e+18	اكبر حدث عالمي اكسبو دبي حق العالم	4/3/22 23:26	0	0	101	4/3/2022	8:26:59 PM	1
3404	1.510710e+18	جمال اكسبو وجمال دبي	4/3/22 23:07	0	0	38	4/3/2022	8:07:49 PM	1
3405	1.510710e+18	...مشكورين بارك اله فيكم نتمنا نرا اكسبو الملكة	4/3/22 23:06	8	0	187	4/3/2022	8:06:03 PM	1
3406	1.510710e+18	وداعا اكسبو دبي دولة زاءر راكب طرق دبي ...	4/3/22 23:04	0	0	273	4/3/2022	8:04:27 PM	1
3407	1.510700e+18	فعلا دبي استثناءية المعرض الهندي جوهم اكسبو	4/3/22 22:23	0	0	91	4/3/2022	7:23:32 PM	1
3408	1.510690e+18	...اختام معرض اكسبو دبي يومين الخميس فرصة المعرض	4/3/22 21:38	1	0	304	4/3/2022	6:38:24 PM	1
3409	1.510690e+18	...الحمدله انجاز راءع شكرا الامارات الحبيبه نبارك	4/3/22 21:30	0	2	160	4/3/2022	6:30:47 PM	1
3410	1.510690e+18	...اكسبو ابرز الارقام القياسية حقها جناح السعودية	4/3/22 21:30	0	0	100	4/3/2022	6:30:06 PM	1

Figure 37 display the data frame after conversion

Now we will split our dataset to two variables, X will contain the features – our raw tweets and y will contain our target variables which can be 0, 1. Then we will need to transform the tweets into a representation that can be digested by the model. Since we have a class imbalance – One of the classes is underrepresented in our case the negative class.We had to do this part differently we will have two training and testing data sets one for our raw data (that is imbalanced) and one for our over sampled data which will use resampling techniques to ensure the model trains on both classes. Despite fixing the class imbalance it is still a problem since resampling will not equate to having more raw 'negative' tweets. First, we will show how we split the raw data.

```python
X = tweets['Tweet']  # features
y = tweets['Class'] # classes
```
[9] ✓ 0.8s Python

```python
# Convert to a vector representation
unsampled_tfidf = TfidfVectorizer()
unsampled_X = unsampled_tfidf.fit_transform(X)
```
[10] ✓ 0.2s Python

Figure 38 splitting and converting the features

Our features ended up being **14747** and the number of data we have after dropping the neutral class is **3411**.

```
unsampled_X
[11] ✓ 0.4s

<3411x14747 sparse matrix of type '<class 'numpy.float64'>'
	with 53393 stored elements in Compressed Sparse Row format>
```

Figure 39 displaying the features

Splitting the Features

We can now start preparing the data for training, we split the data and target classes into training and testing sets. Since our dataset is insufficient we could not add a validation set. The split was **70%** training and **30%** testing.

```
X_train_unsampled, X_test_unsampled, y_train_unsampled, y_test_unsampled = train_test_split(unsampled_X, y, test_size=0.3, random_state=27)
[12] ✓ 0.2s

print("Training set has {} samples.".format(X_train_unsampled.shape[0]))
print("Testing set has {} samples.".format(X_test_unsampled.shape[0]))
[13] ✓ 0.2s

Training set has 2387 samples.
Testing set has 1024 samples.
```

Figure 40 splitting the raw data into sets

Now, to combat class imbalance we split the original data into up sampled training and testing sets. Then we performed vectorization afterwards.

```
upsampled_tfidf = TfidfVectorizer()
upsampled_tfidf.fit(X)
[14] ✓ 0.3s                                                    Python

TfidfVectorizer()

X_train_upsampled, X_test_upsampled, y_train_upsampled, y_test_upsampled = train_test_split(X, y, test_size=0.3, random_state=27)
[15] ✓ 0.1s                                                    Python
```

Figure 41 preparing for up sampling

```
X_train_upsampled = upsampled_tfidf.transform(X_train_upsampled)
X_test_upsampled = upsampled_tfidf.transform(X_test_upsampled)
upsampled_X = upsampled_tfidf.transform(X)
[17] ✓ 0.2s                                                    Python

X_train_upsampled
[18] ✓ 0.4s                                                    Python

<2387x14747 sparse matrix of type '<class 'numpy.float64'>'
	with 37175 stored elements in Compressed Sparse Row format>
```

Figure 42 transforming the data to be up sampled

Now we will use random over sampler to fit and resample our training data. This way the model trains on equal numbers of both classes. The function works by randomly duplicating 'minority' class entries, until it reaches the number of majority class entries.

```
ros = RandomOverSampler(random_state=27)

X_train_upsampled, y_train_upsampled = ros.fit_resample(X_train_upsampled, y_train_upsampled)
```
[19] ✓ 0.1s

Figure 43 resampling the training dataset

The following figure illustrates the benefits of this techniques. notice how low the negative class before up sampling.

```
print("Data before upsampling: {} samples.".format(y_train_unsampled.value_counts()[0]))
print("Data before upsampling: {} samples.".format(y_train_unsampled.value_counts()[1]))
```
[39] ✓ 0.6s

```
Data before upsampling: 148 samples.
Data before upsampling: 2239 samples.
```

```
print("Data after upsampling: {} samples.".format(y_train_upsampled.value_counts()[0]))
print("Data after upsampling: {} samples.".format(y_train_upsampled.value_counts()[1]))
```
[38] ✓ 0.1s

```
Data after upsampling: 2239 samples.
Data after upsampling: 2239 samples.
```

Figure 44 difference between the raw and sampled datasets

Training Pipeline

The pipeline allows us to reduce any repeated code, in the beginning of this section we explained the criteria models are evaluated based on. In the pipeline we first take in the classifier, training and testing datasets and the original dataset.

We first train the classifier using the *.fit* method from sklearn. we also clock in the time it took to train the classifier, next we predict the testing set and a portion of the training set (this was optional) and also record the time it took to make a prediction. Now that we trained, and stored the times for the classifier. we will move on to scoring it.

We scored the model accuracy given its training data. This measures how will it preformed on the training set. Then we go into more detail by scoring its testing set, and the subset of predicted training data – again the last part is optional.

Since our data is not robust enough we did 10-folds cross validation which takes in the model, X and y datasets.

Finally, we scored the model's f-beta scores for both training and testing. The last part simply prints the accuracies along with their confusion matrices.

```python
def train_predict_pipeline(model, X_train, y_train, X_test, y_test, X, y):
    print("            {} Training            ".format(model.__class__.__name__))
    results = {}

    start = time()  # Training start
    model = model.fit(X_train, y_train) # Train the model
    end = time()  # Training end
    results['training_time'] = end - start # Store the time

    start = time()  # Prediction start
    predictions_test = model.predict(X_test) # Predict
    predictions_train = model.predict(X_train[:300])
    end = time()  # Prediction end
    results['prediction_time'] = end - start # Store the time


    results['model_accuracy'] = model.score(X_train, y_train) # Overall accuracy

    # Cross validation score
    cross_validation_scores = cross_val_score(model,X, y, cv=10)
    results['model_cross_validation'] = np.mean(cross_validation_scores)

    # Accuracy scores - for plotting
    results['accuracy_train'] = accuracy_score(y_train[:300], predictions_train)
    results['accuracy_test'] = accuracy_score(y_test, predictions_test)

    # F-scores
    results['fbeta_train'] = fbeta_score(y_train[:300], predictions_train, beta=0.5)
    results['fbeta_test'] = fbeta_score(y_test, predictions_test, beta=0.5)

    # Print the report
    print('            Accuracy Report            ')
    print('Model Accuracy: %.2f' % results['model_accuracy'])
    print('10-Fold Cross Validation: %.2f' % results['model_cross_validation'])
    print('F-beta Score (Training): %.2f' % results['fbeta_train'])
    print('F-beta Score (Testing): %.2f' % results['fbeta_test'])
    print('            Confusion Matrix            ')
    print(confusion_matrix(y_test, predictions_test))
    print(classification_report(y_test, predictions_test))

    display = ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, display_labels=['negative', 'positive'], cmap=plt.cm.Blues)
    display.ax_.set_title('Confusion Matrix Display')
    plt.show()

    # Return the results and the classifier
    return results, model
```

Figure 45 training pipeline

Now we can create our initial classifiers with basic hyperparameters.

```python
SVC_classifier = SVC(random_state=0, probability=True)
AdaBoost_classifier = AdaBoostClassifier(random_state=0)
Naivebayes_classifier = BernoulliNB()
```

✓ 0.1s

Figure 46 create the classifiers

And use the classifiers with our pipeline.

```python
results = {}

for classifier in [SVC_classifier, AdaBoost_classifier, Naivebayes_classifier]:
    classifier_name = classifier.__class__.__name__
    results[classifier_name] = {}
    results[classifier_name], classifier = train_predict_pipeline(
        classifier, X_train_upsampled, y_train_upsampled, X_test_upsampled, y_test_upsampled, upsampled_X, y)
```

✓ 1m 53.4s

Figure 47 calling the pipeline

Output for SVM – it perfectly identifies all positive classes but completely disregards negative ones.

```
            SVC Training
            Accuracy Report
Model Accuracy: 1.00
10-Fold Cross Validation: 0.94
F-beta Score (Training): 1.00
F-beta Score (Testing): 0.94
            Confusion Matrix
[[  4  70]
 [  1 949]]
              precision    recall  f1-score   support

           0       0.80      0.05      0.10        74
           1       0.93      1.00      0.96       950

    accuracy                           0.93      1024
   macro avg       0.87      0.53      0.53      1024
weighted avg       0.92      0.93      0.90      1024
```

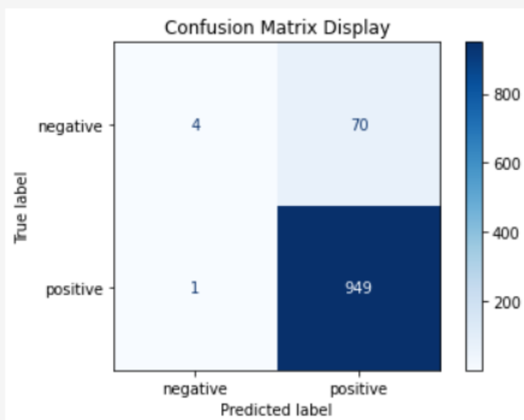


Figure 48 SVM pipeline output

Output for AdaBoost – Performs worse than SVC in terms of positive classes but better with negative ones

```
            AdaBoostClassifier Training
            Accuracy Report
Model Accuracy: 0.90
10-Fold Cross Validation: 0.94
F-beta Score (Training): 0.93
F-beta Score (Testing): 0.95
            Confusion Matrix
[[ 53  21]
 [146 804]]
              precision    recall  f1-score   support

           0       0.27      0.72      0.39        74
           1       0.97      0.85      0.91       950

    accuracy                           0.84      1024
   macro avg       0.62      0.78      0.65      1024
weighted avg       0.92      0.84      0.87      1024
```

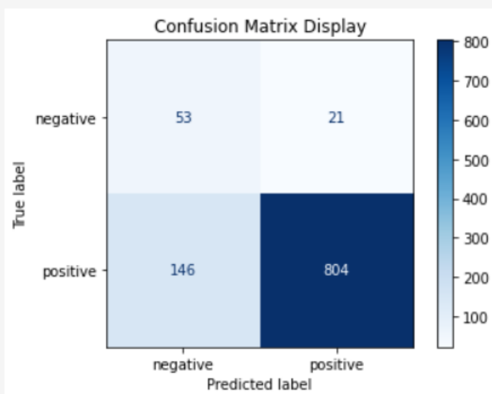


Figure 49 AdaBoost pipeline output

Output for Naïve Bayes – performs the same as AdaBoost when it comes to negative classes. But better in positive ones.

```
            Accuracy Report
Model Accuracy: 0.99
10-Fold Cross Validation: 0.93
F-beta Score (Training): 1.00
F-beta Score (Testing): 0.97
            Confusion Matrix
[[ 53  21]
 [ 61 889]]
              precision    recall  f1-score   support

           0       0.46      0.72      0.56        74
           1       0.98      0.94      0.96       950

    accuracy                           0.92      1024
   macro avg       0.72      0.83      0.76      1024
weighted avg       0.94      0.92      0.93      1024
```

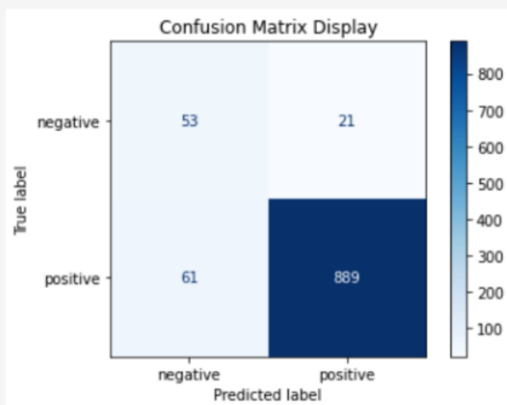


Figure 50 Naïve Bayes pipeline output

Comparing Up sampled vs Raw Datasets

Here we showcase the difference when the model trains on imbalanced data.

SVC – left is the raw and right is the up sampled

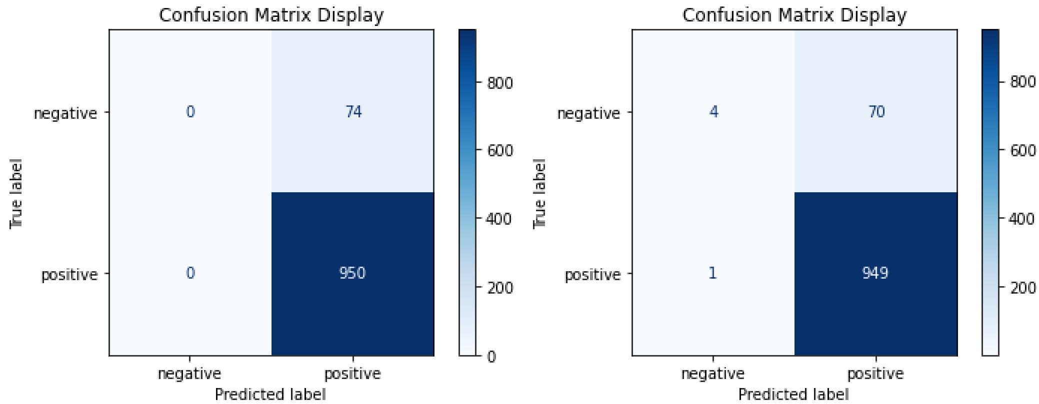


Figure 51 SVC comparison

AdaBoost – left is the raw and right is the up sampled

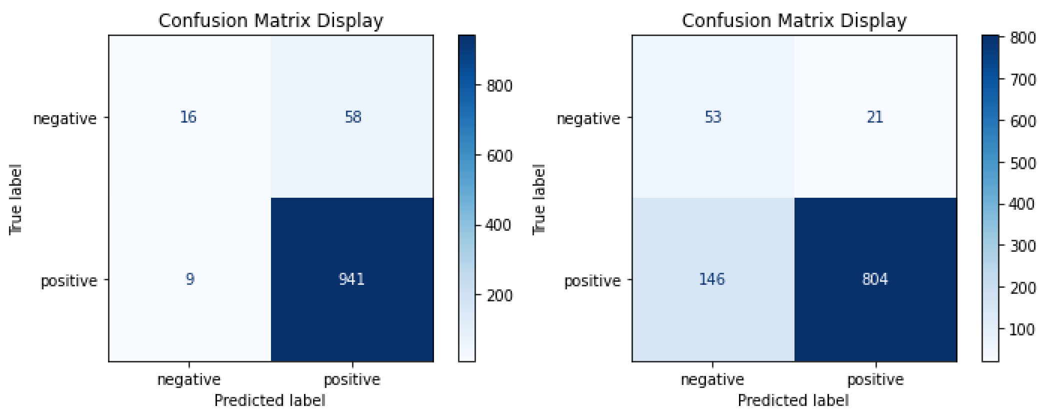


Figure 52 AdaBoost comparison

Naïve Bayes – left is the raw and right is the up sampled

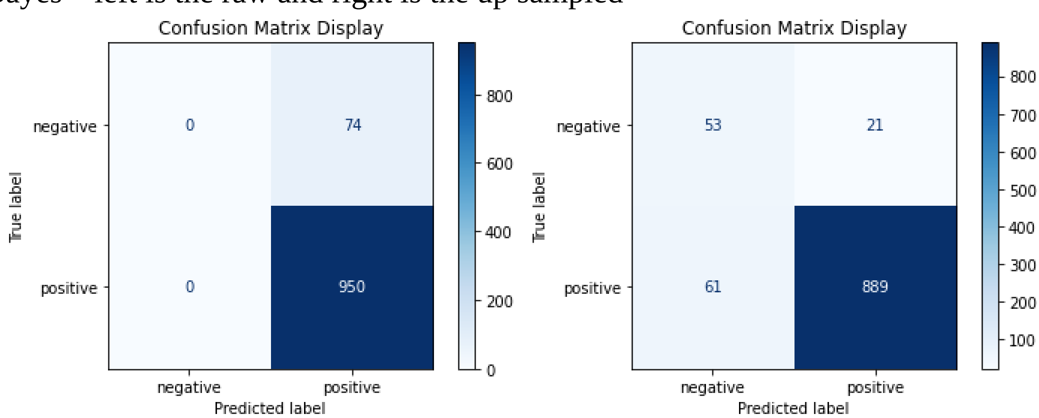


Figure 53 Naive bayes comparison

Model Evaluation

Now we will evaluate which model to choose, the best way to do so is to visually map out our scoring criteria. The following function does so for three classifiers

```
def evaluate(results, accuracy, f1):
    """
    Visualization code to display results of various learners.

    inputs:
      - learners: a list of supervised learners
      - stats: a list of dictionaries of the statistic results from 'train_predict()'
      - accuracy: The score for the naive predictor
      - f1: The score for the naive predictor
    """

    # Create figure
    fig, ax = plt.subplots(2, 3, figsize=(15, 10))

 # Constants
    bar_width = 0.3
    colors = ['#083471', '#1F6EB3', '#56A0CE']

    for k, learner in enumerate(results.keys()):
        for j, metric in enumerate(['training_time', 'accuracy_train', 'fbeta_train', 'prediction_time', 'accuracy_test', 'fbeta_test']):
            ax[j//3, j % 3].bar(k*bar_width, results[learner]
                                [metric], width=bar_width, color=colors[k])

    # Add unique y-labels
    ax[0, 0].set_ylabel("Time (in seconds)")
    ax[0, 1].set_ylabel("Accuracy Score")
    ax[0, 2].set_ylabel("F-score")
    ax[1, 0].set_ylabel("Time (in seconds)")
    ax[1, 1].set_ylabel("Accuracy Score")
    ax[1, 2].set_ylabel("F-score")

    # Add titles
    ax[0, 0].set_title("Model Training")
    ax[0, 1].set_title("Accuracy Score on Training Subset")
    ax[0, 2].set_title("F-score on Training Subset")
    ax[1, 0].set_title("Model Predicting")
    ax[1, 1].set_title("Accuracy Score on Testing Set")
    ax[1, 2].set_title("F-score on Testing Set")

    # Add horizontal lines for naive predictors
    ax[0, 1].axhline(y=accuracy, xmin=-0.1, xmax=3.0,
                     linewidth=1, color='k', linestyle='dashed')
    ax[1, 1].axhline(y=accuracy, xmin=-0.1, xmax=3.0,
                     linewidth=1, color='k', linestyle='dashed')
    ax[0, 2].axhline(y=f1, xmin=-0.1, xmax=3.0, linewidth=1,
                     color='k', linestyle='dashed')
    ax[1, 2].axhline(y=f1, xmin=-0.1, xmax=3.0, linewidth=1,
                     color='k', linestyle='dashed')

    # Set y-limits for score panels
    ax[0, 1].set_ylim((0, 1))
    ax[0, 2].set_ylim((0, 1))
    ax[1, 1].set_ylim((0, 1))
    ax[1, 2].set_ylim((0, 1))

    # Create patches for the legend
    patches = []
    for i, learner in enumerate(results.keys()):
        patches.append(mpatches.Patch(color=colors[i], label=learner))
    plt.legend(handles=patches, bbox_to_anchor=(-.80, 2.53),
               loc='upper center', borderaxespad=0., ncol=3, fontsize='x-large')

    # Aesthetics
    plt.suptitle(
        "Performance Metrics for Three Supervised Learning Models", fontsize=16, y=1.10)
    plt.show()
```

Figure 54 Evaluation function

Now we can call the function, and as we can see we supplied the function with 0.5. we used this as the random case. A poor model will most likely get accuracy scores around 50%. We can clearly see that in terms of time the Naïve Bayes is the best which is one of its advantages. and the AdaBoost had consistent results in both training and testing since it is not prone to overfitting. SVC did take much longer than the other two models.

```
evaluate(results, 0.5, 0.5)
```

Figure 55 call the plotting function

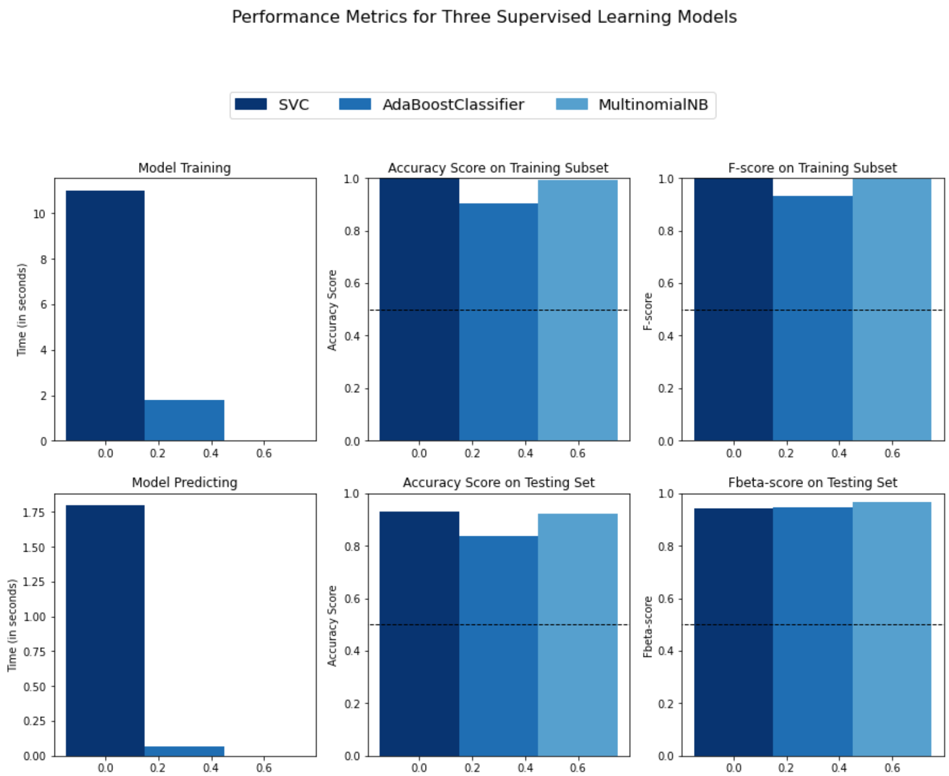


Figure 56 performance evaluation

We also plotted the ROC curve which takes into consideration the true positive rates and the false positive rates. It also plotted the AUC score. AdaBoost underperformed significantly worse than SVC and Naïve Bayes.

```
svc_display = plot_roc_curve(SVC_classifier, X_test_upsampled, y_test_upsampled)
ada_display = plot_roc_curve(AdaBoost_classifier, X_test_upsampled, y_test_upsampled, ax=svc_display.ax_)
naive_display = plot_roc_curve(Naivebayes_classifier, X_test_upsampled, y_test_upsampled, ax=ada_display.ax_)
naive_display.figure_.suptitle("ROC curve comparison")

with warnings.catch_warnings():
    warnings.simplefilter("ignore")

plt.show()
```

Figure 57 ROC and AUC

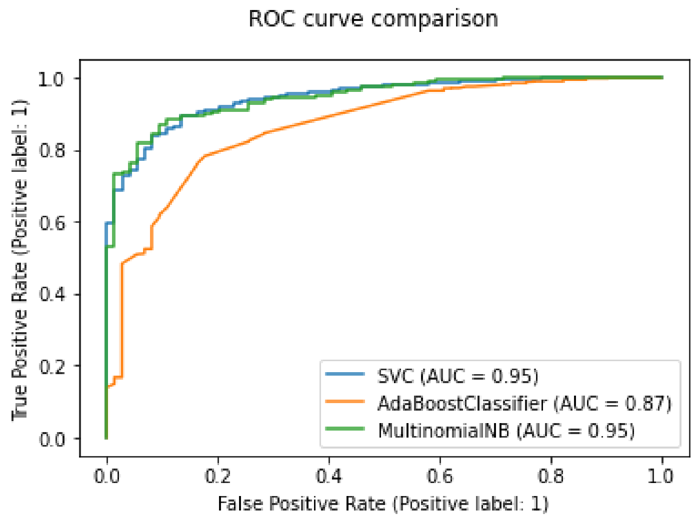


Figure 58 ROC curve chart

Fine Tuning

Finally, we can fine tune the model's parameters. We selected SVC since it had high accuracies and multiple parameters to be fine-tuned. We will use grid search method by sklearn. It works by allowing us to set multiple hyperparameters and fit the model to every possible combination. Then it takes the best fit parameters and trains the model based on it.

The C parameter is the soft/hard margin we discussed. and depending on your data distribution one may yield better results than the other.

```
parameters = {'C': [0.1, 0.5, 1, 1.2, 1.5, 2, 5],
              'kernel': ['linear']}
```

Figure 59 our chosen parameters

Next, we create a scorer to compare the fine-tuned model. and initialize the grid by our classifier, parameters and scorer and train it.

```
scorer = make_scorer(fbeta_score, beta=0.5)
grid = GridSearchCV(SVC(random_state=0), param_grid=parameters, scoring=scorer ,refit=True, verbose=3)
```
✓ 0.4s

```
gridSVC = grid.fit(X_train_upsampled, y_train_upsampled)
```
14.6s

Figure 60 training using the grid search

Comparing the results, we managed to only increase the accuracy and f-beta scores by 1%

```python
gridSVC.best_estimator_
```

```
SVC(C=1, kernel='linear', random_state=0)
```

```python
# Original SVC accuracy
results['SVC']['accuracy_test']
```

```
0.9306640625
```

```python
# Original SVC fbeta score
results['SVC']['fbeta_test']
```

```
0.944090728213291
```

```python
# Fine tuned SVC accuracy
accuracy_score(y_test_upsampled, gridSVC.predict(X_test_upsampled))
```

```
0.939453125
```

```python
# Fine tuned SVC fbeta score
fbeta_score(y_test_upsampled, gridSVC.predict(X_test_upsampled), beta=0.5)
```

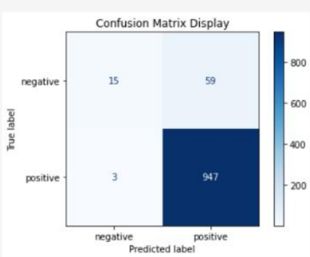
```
0.9519501407318053
```

Figure 61 accuracy scores comparisons

Finally, using our pipeline we can see visually how it improved especially in terms of predicting negative classes.

```python
train_predict_pipeline(SVC(C=1, kernel='linear', random_state=0), X_train_upsampled, y_train_upsampled, X_test_upsampled, y_test_upsampled, upsampled_X, y)
```

```
SVC Training
```




```
({'training_time': 0.9723687171936035,
  'prediction_time': 0.9728827476501465,
  'model_accuracy': 0.9993300580616347,
  'model_cross_validation': 0.943413764126837,
  'accuracy_train': 0.9993300580616347,
  'accuracy_test': 0.939453125,
  'fbeta_train': 0.9997317356702137,
  'fbeta_test': 0.9519501407318053},
 SVC(C=1, kernel='linear', random_state=0))
```

Figure 62 final plotting of the optimized model

CONCLUSION

The biggest challenges we faced were dataset related. Once Expo ended on March 31st, we have gotten significantly less data. Despite that, our data consisted of mostly ads, news and unrelated tweets to the topic. which lead the dataset to decrease from around 7000 entries to 5000 and when we dropped the neutral class, we lost an extra 2000.

Another challenge relating to the dataset was the class imbalance, approximately 7% of the data was negative and 93% was positive. This caused a problem where the model would disregard any negative data into being positive – and led to high accuracies. Nonetheless we attempted to deal and correct each problem we encountered.

Tools Used

These are the tools and libraries we have used during this phase. We will briefly describe them and what they were used in.

NumPy

A library used to transform and manipulate multi-dimensional arrays. it also has mathematical expressions support.

PANDAS

A Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labelled” data both easy and intuitive [1]. Used to store the *.csv* file data in a convenient and easy to process format. Also, used various functions from the library to copy, drop and assess our dataset [8].

Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible [9].

Scikit Learn

Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms. It’s built upon some of the technology we might already be familiar with, like NumPy, pandas, and Matplotlib [10].

Imbalanced Learn

A library build upon sklearn. in which deals with imbalanced data and provides many sampling solutions.

Seaborn

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics [11].

References

1| S. Gupta, "Sentiment Analysis: Concept, Analysis and Applications", Medium, 2017. [Online]. Available: https://towardsdatascience.com/sentiment-analysis-concept-analysis-andapplications-6c94d6f58c17 (accessed: April 15, 2022).

2| Mazajak: An Online Arabic Sentiment Analyser. Ibrahim Abu Farha and Walid Magdy. In Proceedings of the Fourth Arabic Natural Language Processing Workshop (WANLP). 2019.

3| Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

4| Ayush Singh Rawat, “What is Descriptive Analysis?- Types and Advantages | Analytics Steps,” *AnalyticsSteps*, 2021. https://www.analyticssteps.com/blogs/overview-descriptive-analysis (accessed Apr. 16, 2022).

5| “What Is Machine Learning: Definition, Types, Applications and Examples,” *Potentia Analytics*, Dec. 19, 2019. https://www.potentiaco.com/what-is-machine-learning-definition-types-applications-and-examples/#:~:text=These%20are%20three%20types%20of,unsupervised%20learning%2C%20and%20reinforcement%20learning. (accessed Apr. 27, 2022).

6| https://www.facebook.com/MachineLearningMastery, “A Gentle Introduction to Model Selection for Machine Learning,” *Machine Learning Mastery*, Dec. 2019. https://machinelearningmastery.com/a-gentle-introduction-to-model-selection-for-machine-learning/ (accessed Apr. 27, 2022).

7| “1.9. Naive Bayes,” *scikit-learn*, 2022. https://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes (accessed Apr. 27, 2022).

8| "pandas - Python Data Analysis Library," Pydata.org, 2022. [Online]. Available: https://pandas.pydata.org/. [Accessed 7 March 2022].

9| 2022. [online] Available at: <https://matplotlib.org/> [Accessed 27 April 2022].

10| Codecademy. 2022. What is Scikit-Learn? | Codecademy. [online] Available at: <https://www.codecademy.com/article/scikit-learn> [Accessed 27 April 2022].

11| Seaborn.pydata.org. 2022. seaborn: statistical data visualization — seaborn 0.11.2 documentation. [online] Available at: <https://seaborn.pydata.org/> [Accessed 27 April 2022].

APPENDICES

FILES MAPPING

Table 1 files mapping

FILE NAME	DESCRIPTION
Tweets/final_tweets_cleaned.csv	Cleaned dataset from previous phase in .csv format. Used in classification
Tweets/tweets_classified.csv	Mazajak's labelled dataset in .csv format
Tweets/final_tweets_classified.csv	Manually classified dataset in .csv format
mazajak_api	Python methods to use mazajak's api from
tweets-analysis.ipynb	Predictive analysis code
tweets-statistics.ipynb	Descriptive analysis code
tweets-labeling.ipynb	Mazajak's labelling code